**STUDENT NAME:** HUU CUONG NGUYEN

**STUDENT ID:** 44491158

**PROJECT:** HUMAN ACTIVITY RECOGNITION

**PROJECT TITLE:** RADAR (Results and Analysis of Detecting Activities in Real-time)

Our daily lives are associated with many activities and perhaps sometimes, we are asking ourselves: What activities that we have done most in a day or in a week? Nowadays, with the arising of modern technique, we can literally answer that question by having some sensors placed on our bodies to track the activities that we have done throughout a day or a week. This project will help us understand it, which is called human activity recognition, a lot better.

Information technology is a field that has quite many projects that we have been given to get done. For each project, we should sketch out the product backlog and sprint, as well as each sprint burn down chart, to manage our progress when each sprint (each job) is completed. The same advice can be applied to this project of human activity recognition represented in this paper. Here is a concise and general product backlog and sprint for our project:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| User stories | Priority | Task | Week 1 | Week 2 | Week 3 | Week 4 |
| Percentage of work done | | | |
| * As an analyser, I want to firstly read the dataset and then visualize the sensor data so that I can get some idea of the underlying human activity pattern for better future analysis. | 1 | Import all necessary libraries into Python for code writing. | 100% |  |  |  |
| Write code to read the dataset and print out some general information that we want to know about. | 100% |  |  |  |
| Get first and last 5 rows in the dataset to get an overall picture of the human activities. | 100% |  |  |  |
| Get a table containing min, standard deviation. mean values, etc. for future analysis purposes. | 100% |  |  |  |
| * As a system developer, I want to apply signal filtering to my product and then visualize the cleaned data so that I can remove some unwanted components (e.g. noise) or features from a signal and get a more exact and overall picture of the activities that the subjects have done. | 2 | Get the desired values of one of the 13 activities and sketch the plot for those data for the desired axes (gyroscope axes or accelerometer axes). | 10% | 90% |  |  |
| Apply signal filtering to the plot of those data to remove unwanted signal. | 0% | 100% |  |  |
| * As a system developer, I want to extract a lot of features from many sensors so that I can get just the required and necessary values to analyse what the sensors can do in recognizing human activities and get an understanding and knowledge of how accurately the sensors can detect human activity. | 3 | Read the dataset of all the desired number of participants, say, 10. | 0% | 100% |  |  |
| Get the values of all 13 activities and apply Butterworth signal filtering to each of them. | 0% | 100% |  |  |
| Set the desired percentage of the dataset to use for training and testing and decide the number of training and testing samples. | 0% | 10% | 90% |  |
| Create the sample data for training for many of the axes and sensors and extract desired features (e.g. standard deviation, median, min, etc.), and store those features in the training dataset and deliver the results to the created file called training\_data.csv. | 0% | 0% | 100% |  |
| Do the same as creating the training dataset for testing data and deliver the results to the created file called testing\_data.csv. | 0% | 0% | 100% |  |
| * As a system tester, I want to evaluate and test the application after training model so that I can accurately measure the performance of a trained model, test how accurate the sensors are, and fix errors if any. | 4 | Read the training and testing data in the files training\_data.csv and testing\_data.csv. | 0% | 0% | 100% |  |
| Prepare the values for training and testing input, and construct feature normalization. | 0% | 0% | 100% |  |
| Learn activity recognition by building KNN classifier, then calculate the classification accuracy and get the confusion matrix to view the classification for each activity. Use the GridSearch to find the optimal classifier for KNN based on the tuned parameters and accuracy score passed. Get the accuracy score and confusion matrix for optimal classifier for KNN. | 0% | 0% | 100% |  |
| Do the same as building KNN classifier for building SVM classifier. | 0% | 0% | 60% | 40% |
| * As a code writer, I want to refactor my code so that I can make those code easy to read, understand and extensible for future development and improvement. | 5 | Comment on codes which are not clear or hard to understand. | 0% | 0% | 0% | 100% |
| Remove all distracted information and codes. | 0% | 0% | 0% | 100% |
| Arrange codes in a readable, attractive style and format. | 0% | 0% | 0% | 100% |

Human activity recognition is the issue of arranging groupings of accelerometer and gyroscope recorded by particular devices or advanced mobile phones into known well-characterized movements. Human Activity Recognition (HAR) is also described as anticipating what an individual is doing based on a hint of their movement utilizing sensors.

Activities are frequently typical indoor exercises, for example, standing, sitting, rope jumping, and going up stairs, etc. Sensors are frequently placed on the person, for example, a cell phone or vest, clothes and regularly record accelerometer and gyroscope data in three measurements (dimensions) (x, y, z). The thought is that once the participant's action is perceived and known, a smart computer system would then be able to offer help.

To begin with, we should divide the project into 5 sprints (as the above table), using the SCRUM Sprint and Design technique, for easy data construct and implementation. The first sprint is about reading and visualizing the sensor data. For this sprint, we need to firstly import all necessary libraries into our project to write some code for reading all the necessary and desired information, and then just get those code executed so that we can get some general information about the data that we are experimenting with to get an overall picture of it. The next sprint is about signal filtering and cleaned data. First of all, we must identify what data of which person that we are interested in and then get those data values by creating some code to read and extract those values. Next, we need to construct the plot of those data values and apply signal filtering to that plot as the plot initially contains some annoying and unwanted noise that we might want to get rid of. When it comes to the third sprint, it is mainly about feature engineering for training and testing data. Feature engineering simply means extracting (getting) some desired features (values) to ‘engineer’ (work on) it to analyse the result and deliver the conclusion for future purposes. We can do this by firstly reading the dataset of the interested number of participants and then based on those information, just get the necessary features and write some code to train and test (examine) data. What’s next, the very last sprint is about evaluation and testing. After we have done all the above jobs, we need to evaluate all the extracted information to determine if this data benefits us and helps us in our experiment and to get some kind of knowledge of how those extracted features affect the human activity recognition. And the final sprint is about refactoring code, which is last but not least. While writing the code, we might not focus on the structure of the code, the way that the code is organized yet, so it easily gets messy and hard to read or maybe at worst, unreadable. Therefore, we should manage it by just putting some comments on how this code does or what can we know from that code, and indenting the codes properly, deleting all unused or unnecessary information, replacing all magic numbers with the meaningful ones, finding ways to best deliver the meaning to the code reader, tester or code developer for future improvement, easy maintenance and fixing if any failures occur.

Now, we are going to focus on analysing the activity recognition and we firstly begin with sprint 1. This sprint initially has 5 hours of work required to be done and it needs to be completed within maximum of 4 days in week 1. To easily keep track on our progress, the burndown chart is sketched out as below:

**SPRINT 1**

Sprint 1 Burndown Chart from 19/08 to 22/08 (in week 1)

It is ideal and estimated to be finished within only 3 days, but in real time, it was done within 4 days as a little bit less effort was devoted on it.

To start with, we need to place sensor nodes on human body to sense the movements and detect 13 activities, which are sitting, lying, standing, washing dishes, vacuuming, sweeping, walking outside, ascending stairs, descending stairs, treadmill running, slow bicycling, fast bicycling, and rope jumping. There is a total of 4 nodes that should be placed on the body. The first one is placed on wrist, the second one is on chest, the third one is on hip, and the last one is on ankle. Each of these nodes has 6 coordinate axes, and in the dataset they are numbered as from 0 to 23 for this total of 24 coordinate axes. The first 6 numbers from 0 to 5 indicate 3 accelerometer axes and 3 gyroscope axes placed on wrist; the next 6 numbers indicate the counterpart placed on chest; next is for hip; and the last 6 numbers are for ankle. Accelerometers are used to recognize the orientation or to sense the speeding up event of objects and can only realize the fact that an object has moved or is moving in a specific way, whereas Gyroscopes measure the angular rate of rotational movement around at least one axis and can measure complex movement precisely in various movements, tracking the position and rotation of a moving object. Here, we have a total of 19 datasets representing 19 different subjects for the experiment.

Initially, we will get general information of all datasets by reading all of them and printing out the number of rows and then choose an arbitrary dataset to read and work on it from now on, say, the dataset number 5, which represents the fifth person. We can write some python code to get an overall picture of the sensor data. In this dataset, which contains 254178 rows, we will get the data of the first 5 rows and the last 5 rows (presented below) partially pointing out the activity of sitting and rope jumping respectively (by looking at the last column, we can know which is for sitting and which is for rope jumping: 1 is for sitting and 13 is for rope jumping).

|  | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **…** | **15** | **16** | **17** | **18** | **19** | **20** | **21** | **22** | **23** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | -0.071637 | -0.104500 | 0.98682 | 1.2854 | -1.163600 | -2.2642 | 0.018673 | -0.66443 | 0.68869 | 4.2520 | ... | -4.4107 | -5.93760 | 2.6961 | -0.83067 | -0.20180 | -0.52616 | -40.131 | 27.299 | -10.359 | 1 |
| **1** | -0.071637 | -0.120530 | 0.99090 | 1.6516 | -0.797450 | -2.2642 | 0.026527 | -0.65231 | 0.69271 | 2.0550 | ... | -3.3122 | -4.47290 | 2.6961 | -0.82267 | -0.17481 | -0.54961 | -40.131 | 27.299 | -10.359 | 1 |
| **2** | -0.059915 | -0.108510 | 0.99090 | 2.3839 | -0.431280 | -2.6304 | 0.010820 | -0.66847 | 0.67260 | 0.5903 | ... | -3.3122 | -2.64210 | 3.0623 | -0.83867 | -0.19409 | -0.55351 | -40.131 | 27.299 | -11.824 | 1 |
| **3** | -0.056007 | -0.100490 | 0.99498 | 2.7501 | -0.431280 | -2.6304 | 0.010820 | -0.67654 | 0.67662 | -1.6067 | ... | -2.2137 | -0.81126 | 2.6961 | -0.83067 | -0.19795 | -0.54961 | -37.202 | 27.299 | -10.359 | 1 |
| **4** | -0.067730 | -0.092478 | 0.99498 | 3.1162 | -0.065115 | -2.6304 | 0.002966 | -0.67250 | 0.66456 | -3.4375 | ... | -2.2137 | 0.65340 | 2.6961 | -0.82267 | -0.19409 | -0.57305 | -37.202 | 28.764 | -10.359 | 1 |

The first 5 rows

|  | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **…** | **15** | **16** | **17** | **18** | **19** | **20** | **21** | **22** | **23** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **254173** | -0.95473 | 0.98971 | -0.14452 | -96.481 | 130.29 | 240.50 | 0.144330 | -3.9560 | 0.62034 | -658.51 | … | 164.030 | 314.4600 | 88.379 | -2.0632 | 0.939400 | -1.71000 | -34.272 | 58.057 | -209.55 | 13 |
| **254174** | -0.99771 | 1.12200 | -0.15677 | -95.016 | 128.46 | 224.39 | 0.234650 | -5.0828 | 0.83745 | -662.54 | … | 513.350 | 731.8900 | 197.130 | -3.7921 | -0.487100 | -0.24095 | -54.778 | 66.845 | -178.80 | 13 |
| **254175** | -1.04460 | 1.27030 | -0.16086 | -95.382 | 125.90 | 206.45 | 0.305330 | -6.4438 | 1.05050 | -662.90 | … | 491.380 | 731.8900 | -17.443 | -4.1682 | -0.456260 | 0.30213 | -32.808 | -16.640 | -158.29 | 13 |
| **254176** | -1.08760 | 1.42660 | -0.21395 | -98.312 | 122.97 | 187.41 | 0.073648 | -7.7847 | 1.67370 | -566.60 | … | -68.856 | 477.4000 | -165.740 | -3.1558 | 0.010245 | -0.48709 | -31.343 | -54.722 | -134.86 | 13 |
| **254177** | -1.11490 | 1.58690 | -0.21395 | -104.900 | 120.77 | 165.44 | -0.888410 | -7.7443 | 1.25560 | -210.69 | … | -337.260 | 2.4842 | -151.090 | -2.2753 | 0.515300 | -0.67072 | -50.384 | -38.610 | -104.10 | 13 |

The last 5 rows

We also can know the min, max, standard deviation, etc. values extracted from these sensor nodes by using numpy, which is pretty much like library in python, to write some code (as given in the provided file) and as a result, get the table below:

|  | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **…** | **15** | **16** | **17** | **18** | **19** | **20** | **21** | **22** | **23** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **count** | 254178.000000 | 254178.000000 | 254178.000000 | 254178.000000 | 254178.000000 | 254178.000000 | 254178.000000 | 254178.000000 | 254178.000000 | 254178.000000 | ... | 254178.000000 | 254178.000000 | 254178.000000 | 254178.000000 | 254178.000000 | 254178.000000 | 254178.000000 | 254178.000000 | 254178.000000 | 254178.000000 |
| **mean** | -0.609215 | 0.296369 | 0.376587 | 1.221395 | 1.995649 | 0.364746 | -0.033332 | -0.828625 | -0.106726 | 0.494597 | ... | 0.247012 | -0.274906 | 0.609084 | -1.126444 | 0.087232 | 0.113978 | -2.159677 | 3.821556 | -1.567319 | 7.192707 |
| **std** | 0.544701 | 0.644044 | 0.392859 | 67.576983 | 37.886099 | 63.693531 | 0.164567 | 0.592303 | 0.511935 | 72.024028 | ... | 34.476071 | 42.696717 | 28.111713 | 0.884566 | 1.085362 | 0.541106 | 44.015028 | 62.072057 | 151.081813 | 3.286212 |
| **min** | -4.819200 | -3.423200 | -1.933400 | -646.830000 | -294.830000 | -503.550000 | -2.706500 | -8.422800 | -2.511700 | -662.900000 | ... | -669.730000 | -620.730000 | -701.810000 | -8.086100 | -7.989700 | -6.836000 | -652.360000 | -819.280000 | -685.570000 | 1.000000 |
| **25%** | -0.872670 | -0.064421 | 0.084199 | -16.657000 | -10.684000 | -11.418000 | -0.103060 | -0.979450 | -0.509450 | -10.395000 | ... | -11.368000 | -6.669900 | -8.655000 | -1.402900 | -0.197950 | -0.026064 | -6.443600 | -7.852500 | -17.682000 | 5.000000 |
| **50%** | -0.685110 | 0.051814 | 0.312920 | 0.186890 | -0.065115 | 0.298940 | -0.028448 | -0.854250 | -0.075224 | 0.590300 | ... | 0.715580 | -0.445090 | 0.499110 | -1.010800 | -0.020598 | 0.110680 | 0.879720 | 0.935530 | -0.106480 | 7.000000 |
| **75%** | -0.313900 | 0.564850 | 0.729510 | 13.735000 | 12.018000 | 12.016000 | 0.034380 | -0.628080 | 0.234360 | 13.040000 | ... | 13.165000 | 6.512100 | 7.456300 | -0.810660 | 0.199160 | 0.270870 | 6.738400 | 14.118000 | 36.510000 | 10.000000 |
| **max** | 1.901600 | 5.158100 | 8.542700 | 736.550000 | 315.200000 | 459.470000 | 1.365600 | 3.669100 | 4.592700 | 732.190000 | ... | 734.150000 | 733.350000 | 337.010000 | 8.169600 | 7.721100 | 8.127900 | 432.960000 | 425.690000 | 713.190000 | 13.000000 |

The negative numbers in the above table point out that the subjects do not actually perform the activity in the corresponding column, whereas the positive numbers have the opposite meaning. For example, the negative value at the ‘mean’ row and first column symbolizes that for an average number of times recorded, this participant is not sitting at all or perhaps has some movements while sitting. Conversely, the positive number at the ‘max’ row and first column shows the highest value for the sitting activity, or saying more simply, the greater the positive number is, the higher the possibility that this person is sitting still.

Next, let’s move on to sprint 2, which is signal filtering. Sprint 2 burndown chart below will help us manage this progress more easily:

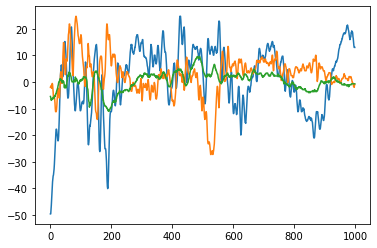
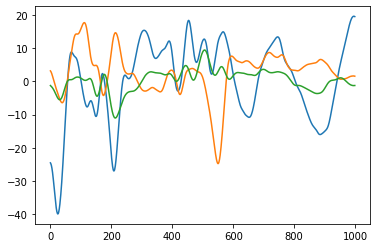
**SPRINT 2**

Sprint 2 Burndown Chart from 24/08 to 29/08 (in week 1 and week 2)

As it can be seen from this chart, the second sprint requires much more effort and time to get done as it involves more tasks than the first one. Specifically, a total of 10 hours needs to be spent on this sprint. One more time, in reality, less effort than expected was spent on this sprint as it required 6 days to get completed while it was supposed to be finished within 5 days.

To analyse the results, it’s quite difficult for us to look at the table to deliver the final outcomes or decisions for future improvements, so we need to carry out some plots of data based on the provided information. For instance, we can get all the sensor data rows for activity 3 (standing) and plot the gyroscope axes G1, G2, G3 that are placed on chest in the time period [500, 1500] by writing some piece of code as given in the file; or we can get a plot of just one axis, say, accelerometer axis A3 placed on ankle, of any activity, for example, sweeping (activity 6), in all time period by changing some numbers in the code which is also shown in the file. What needs to be considered here is that there are some unwanted components (e.g. noise) in the signal preventing us from accurately finding out the information of the subject’s activities. Thus, we should apply signal filtering to our plot to remove those components so that from there we can more effectively visualize it. Here, we use Butterworth technique with the low pass filter. The Butterworth filter is a kind of signal filtering aiming at producing frequency as flat as possible, and is also denoted as maximally flat magnitude filtering technique.

We will have a quick look at the plot of standing for 3 gyroscope axes in the chest sensor before and after applying signal filtering that the code written delivers to us to better understand it:



Before After

We can easily realize that it’s much more readable and simpler to get exact information after we have applied signal filtering. Looking at the plot, we can somehow know what this person mostly does in the time period [500, 1500] in terms of standing. For example, the positive number on the vertical axis, which labels the frequency, in the plot points out that this person is standing in some certain time period, which is represented on the horizontal axis, as the chest sensor might detect that there is no or just a little bit movement of chest in that time period, while the negative number shows that he or she is not standing, which means he or she is moving, in some certain time period.

The more positive or negative the number is, the more chances that this person is standing still or moving a lot.

We have recently done the first 2 sprints and now we keep moving to the third sprint, which is feature engineering. We need to construct a burndown chart for this sprint as well.

**SPRINT 3**

Sprint 3 Burndown Chart from 31/08 to 05/09 (in week 2 and week 3)

For the third sprint to be done, we just need to spend 8 hours of work on it, and this is spread throughout 6 days.

For this job, first of all, we need to extract some necessary features of the desired number of subjects for analysis purposes. Firstly, we might want to read the data of 19 participants and get the values of their 13 activities and apply signal filtering, for the 4 sensors, with Butterworth technique and low pass filter to each of the activities in order to clear, remove unwanted noise and components from the signal. Next, we consider taking 80% of data for training and leaving 20% of it for testing. Now, we will define each period of time containing 1000 different data points and decide the number of feature samples for training and testing. And from here, we can extract a lot of features for training and testing. For instance, we can extract the min, max and mean values for the first 3 accelerometer data in the wrist sensor; or we are also able to extract the standard deviation, min, and median values for the 3 gyroscope data in the chest sensor. At first, we construct the sample data and extract some desired features for training. Here for training, we will extract min, median and max values for the 3 gyroscope data in the hip sensor, ranging from 15 to 18 (exclusive). In total, we get 9 features and 1 label for training. Next, we should do the same for testing or we could extract different features for both training and testing, for example, min, mean and standard deviation for the 3 accelerometer axes A1, A2, A3 in the ankle sensor. Finally, the code written ‘frames’ (creates) the data and produces the training and testing data files and saves them in the same repository.

And the final thing in analysing the activity recognition is the very last sprint which is evaluating and testing the application after having trained the model.

**SPRINT 4**

Sprint 4 Burndown Chart from 06/09 to 10/09 (in week 3 and week 4)

This is the sprint that involves much more tasks to do compared to the rest, so it needs 12 hours of work to get done. Now, the actual work kept up with the expected one but not with the ideal one as it was a day later than the ideal one.

Here, firstly, we should read the training and testing data files that the above steps and code provide to us. Then, we just get the values of the last column, which represents the number of each activity, in the table, so the result will be from 1 to 13. However, in sklearn, the labels should start from 0, so we should subtract 1 from it. Next, we should convert those values to integer type as they are initially represented as floating point numbers. After that, we need to drop (remove) the last column representing the activities’ numbers in the table, otherwise the model will not fit and loss will show up as NAN, and then get the values of the rest columns for training input. For testing input data, we should perform the same action as we have done for training input. Feature normalization, which is pretty much like a method of standardizing the range of independent variables or features of data, is the next step that we need to get done. Using StandardScaler technique, we can scale original feature to be centered around zero for training and testing values. Next, for evaluation, we may consider using 2 metrics here, which are KNN and SVM classifiers. KNN is just some kind of depicting the idea of similarity (sometimes called distance, proximity, or closeness) with some math knowledge we might have gone through in our childhood— calculating the distance between points on a graph. Whereas **support vector machines (SVMs)**are a set of supervised learning methods used for classification, regression and outliers detection.

We need to, at first, declare the KNN classifier and pass the number of neighbors, which will be used by default and set to 3, as parameter to it. Then, just create the predicted class based on the testing data values and print out the accuracy values and confusion matrix to view the classification for each activity based on KNN with 3 neighbors. By writing and executing the code, we will get the accuracy value and confusion matrix as below:

Accuracy: 0.7053742802303263

[[ 38 14 1 3 0 0 0 0 0 0 0 1 0]

[ 15 35 2 3 1 1 0 0 0 0 0 0 0]

[ 14 10 15 13 0 1 0 0 0 0 1 3 0]

[ 8 3 5 76 1 0 1 0 0 0 4 1 0]

[ 0 0 0 1 23 14 8 3 3 0 3 1 1]

[ 0 0 1 5 14 27 16 7 2 0 4 9 0]

[ 1 0 1 0 10 4 195 0 0 3 0 4 1]

[ 0 0 0 0 3 3 6 25 0 1 1 0 0]

[ 0 0 0 0 4 4 1 0 28 1 0 0 0]

[ 0 0 0 0 0 0 0 0 1 95 0 0 0]

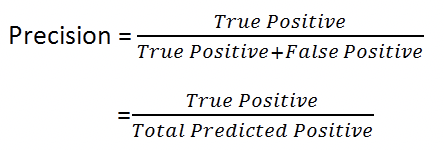
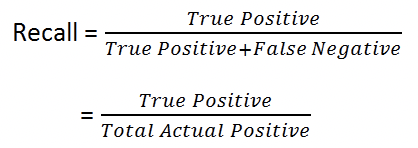
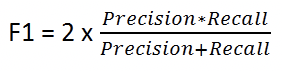
[ 0 1 2 0 1 3 2 0 0 0 78 13 0]

[ 0 0 1 3 1 1 3 0 0 0 18 73 0]

[ 0 0 0 0 3 1 2 0 0 4 0 1 27]]

So, by looking at this matrix, what can we know about it? There is a total of 13 rows and 13 columns in this matrix, so they describe the 13 activities. The columns and the rows stand for the predicted activities and the actual activities that the subjects have done, respectively. From this matrix, we can get the true positives (TP) (cases in which we predicted yes (they perform a particular activity), and they do perform that activity), the false positives (FP) (cases in which we predicted yes, but they don’t actually perform that activity), the true negatives (TN) (cases in which we predicted no, and they don’t perform that activity either), and the false negatives (FN) (cases in which we predicted no, but they do actually perform that activity). For example, for the first activity, which is sitting and represented by the first column (for predicted ones) and the first row (for actual ones), the true positives will be 38, as 38 is the prediction and also the actuality of sitting activity that the subjects have done. The false positives will be 15+14+8+1, which equals to 38, because we can see that on the first column, except the number 38 which is as described before, the rest numbers denote the prediction of sitting activity and the actuality of performing another activities. For instance, the number 15, which is at column index 0 and row index 1, corresponds to the prediction of sitting activity and the actuality of another activity, which, in this case, is lying. The same pattern can be observed for the next number, which is the third number on the first column: 14 stands for the prediction of sitting and actual standing activity, and keeps going until the last number on the first column. Next, just move on to the false negatives. For sitting, we can calculate it based on the first row of the matrix. The value of false negatives will be 14+1+3+1, which equals to 19, as the numbers on the first row, except the number 38, will stand for the prediction of another activity and the actuality of sitting. Here, number 14, which is located at column index 1 and row index 0, characterizes the prediction of lying and actuality of sitting. The same pattern keeps going until the last number on the first row, and then we will sum these numbers up and finally get the value of 19. The last one is the value of true negatives. We can calculate it by combining the rest numbers (other than the ones on the first column and the first row) and then shrink them into a number.

What can we get from these values? Based on these values, we can calculate the precision, recall, F1 and accuracy to fully evaluate the effectiveness of a model. Precision is literally how many of the returned hits were true positives, and is also some kind of ‘how useful the search results are’. Whereas recall literally means how many of the true positives were recalled (found), and is also some kind of ‘how complete the results are’. The F1 score can be described as a weighted average of the precision and recall. And the accuracy measures the possibility of the set of labels predicted for a sample matching the corresponding set of labels. For easy understanding, figuring and visualization, the formulae for these kinds of values are provide below:

page112image2322822960

To have better understanding and more knowledge of these kinds of technique used to evaluate the model, we will calculate some of these values, typically the precision, recall and F1, for all the activities for n\_neighbors set to 3. Consequently, we get the result as presented below:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Sitting | Lying | Standing | Washing  dishes | Vacuuming | Sweeping | Walking  outside | Ascending  stairs | Descending  stairs | Treadmill  running | Bicycling  (50 watt) | Bicycling  (100 watt) | Rope  jumping | Recall  (%) |
| Sitting | 38 | 14 | 1 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 66.67 |
| Lying | 15 | 35 | 2 | 3 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 61.40 |
| Standing | 14 | 10 | 15 | 13 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 3 | 0 | 26.32 |
| Washing  dishes | 8 | 3 | 5 | 76 | 1 | 0 | 1 | 0 | 0 | 0 | 4 | 1 | 0 | 76.77 |
| Vacuuming | 0 | 0 | 0 | 1 | 23 | 14 | 8 | 3 | 3 | 0 | 3 | 1 | 1 | 40.35 |
| Sweeping | 0 | 0 | 1 | 5 | 14 | 27 | 16 | 7 | 2 | 0 | 4 | 9 | 0 | 31.76 |
| Walking  outside | 1 | 0 | 1 | 0 | 10 | 4 | 195 | 0 | 0 | 3 | 0 | 4 | 1 | 89.04 |
| Ascending  stairs | 0 | 0 | 0 | 0 | 3 | 3 | 6 | 25 | 0 | 1 | 1 | 0 | 0 | 64.10 |
| Descending  stairs | 0 | 0 | 0 | 0 | 4 | 4 | 1 | 0 | 28 | 1 | 0 | 0 | 0 | 73.68 |
| Treadmill  running | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 95 | 0 | 0 | 0 | 98.96 |
| Bicycling  (50 watt) | 0 | 1 | 2 | 0 | 1 | 3 | 2 | 0 | 0 | 0 | 78 | 13 | 0 | 78 |
| Bicycling  (100 watt) | 0 | 0 | 1 | 3 | 1 | 1 | 3 | 0 | 0 | 0 | 18 | 73 | 0 | 73 |
| Rope  jumping | 0 | 0 | 0 | 0 | 3 | 1 | 2 | 0 | 0 | 4 | 0 | 1 | 27 | 71.05 |
| Precision  (%) | 50 | 55.56 | 53.57 | 73.08 | 37.70 | 45.76 | 83.33 | 71.43 | 82.35 | 91.35 | 71.56 | 68.87 | 93.10 |  |
| F1 | 57.14 | 58.33 | 35.30 | 74.88 | 38.98 | 37.50 | 86.09 | 67.57 | 77.77 | 95 | 74.64 | 70.87 | 80.59 |  |

As we can see from the above table, the precision of the prediction for the activities performed is quite high for most of the activities and so is the possibility of the true positives that were found (the recall). For precision, the highest value is for treadmill running with over 91%, pointing out that most of the prediction is true in terms of treadmill running, and the smallest one is for vacuuming, illustrating that most of the time the prediction is false with respect to vacuuming. On the other hand, for recall, the greatest number is for treadmill running, indicating there is a high opportunity that the true positives were found, whereas the smallest is for standing, which means that the number of the prediction matching with the actual standing activity constitutes a quite small amount of the overall prediction of all activities when standing is being performed. As a result, all of the above lead to quite large numbers of F1 and accuracy in general.

What’s next, we should get the optimal classifier for KNN by examining tuned parameters, which are the 10 values (from 1 to 10) of the number of neighbors, and accuracy score. After that, we ought to get the best classifier for training and create predicted class for testing based on the optimal classifier that we have got. Here, we will get the optimal classifier having 6 neighbors. Lastly, for KNN, we need to get the accuracy value and the confusion matrix. To prepare for SVM classifier, we should do similarly, but with some different features and values (as specified in the code file).

The last sprint (job) that we need to, or perhaps have to, do is refactoring the code that we have spent time writing.

**SPRINT 5**

Sprint 5 Burndown Chart from 11/09 to 13/09 (in week 4)

The last sprint involves the least number of tasks to do as it is the easiest one among the 5 sprints, so it requires only 3 hours of work being done in 3 days.

In the provided code file, along with each of the easy-to-get-misunderstanding lines of code, there are some comments that will help to clarify it and it will be written in a readable manner. Also, the code is indented properly and some specific tasks are put into particular functions for easier execution, development and more maintainable.

Despite its wonderful applications and usefulness, it does have many challenges and limitations that we, as developers, have to deal with. The first one is subject sensitivity. The accuracy of activity recognition, particularly those based on the accelerometer data, is intensively influenced by the subjects participating in the project. Second, that is location sensitivity. Due to the property of accelerometer in wearable sensors, its raw reading is hugely dependent on the sensors’ orientation and positions on the person’s body. The third one is activity complexity. The movement during the transition period between two activities is tricky for the fundamental classification algorithm to perceive. People doing many activities at the same time may likewise confound the classifier which is trained under one activity-per-segment assumption. What comes next is energy and resource constraints. Persistent sensing as well as online updating of the classification model is required to recognize activity, both of which consume a lot of energy. And also, power consumption is the main factor influencing the size of the battery and sensor nodes. Moreover, insufficient training set is another concern that we also need to consider here. As mentioned before, it is exceptionally desirable that the training data must comprise as many diversities of the participants as could be allowed. However, it is quite hard to coordinate people of different ages and body shapes to gather data under a controlled lab condition, let alone the varieties of the environment itself. Not only does activity recognition present these challenges but also other challenges such as intraclass variability, intraclass similarity, usability, privacy, obtrusiveness, data collection, flexibility, processing and so forth.

In conclusion, this project helps us discover a standard dataset, system architecture, technologies and techniques for human activity recognition. Specifically, we have gained knowledge about how to load and prepare human activity recognition data, how to explore and visualize data for classification in order to generate ideas for modelling and get some underlying pattern of activity recognition and a suite of approaches for framing the problem, preparing the data, modelling, feature engineering, training, testing data and evaluating models for human activity recognition.

**GITHUB VERSION CONTROL**



The Jupyter notebook (Python code) for this project is available on github:

<https://github.com/cuongnguyen44491158/COMP255-Data-Engineering-Assignment>